DOI:10.1145/3107239

Mathematics solves problems by pen and paper. CS helps us to go far beyond that.

BY MARIJN J.H. HEULE AND OLIVER KULLMANN

The Science of Brute Force

RECENT PROGRESS IN automated reasoning and supercomputing gives rise to a new era of brute force. The game changer is "SAT," a disruptive, brute-reasoning technology in industry and science. We illustrate its strength and potential via the proof of the Boolean Pythagorean Triples Problem, a long-standing open problem in Ramsey Theory. This 200TB proof has been constructed completely automatically—paradoxically, in an ingenious way. We welcome these bold new proofs emerging on the horizon, beyond human understanding—both mathematics and industry need them.

Many relevant search problems, from artificial intelligence to combinatorics, explore large search spaces to determine the presence or absence of a certain object. These problems are hard due to combinatorial explosion, and have traditionally been called infeasible. The brute-force method, which at least implicitly explores all possibilities, is a general approach to systematically search through such spaces.

Brute force has long been regarded as suitable only for simple problems. This has changed in the last two decades, due to the progress in Satisfiability (SAT) solving, which by adding brute reason renders brute force into a powerful approach to deal with many problems easily and automatically. Search spaces with far more possibilities than the number of particles in the universe may be completely explored.

SAT solving determines whether a formula in propositional logic has a solution, and its brute reasoning acts in a blind and uninformed way—as a feature, not a bug. We focus on applying SAT to mathematics, as a systematic development of the traditional method of proof by exhaustion.

Can we trust the result of running complicated algorithms on many machines for a long time? The strongest solution is to provide a proof, which is also needed to show correctness of highly complex systems, which are everywhere, from finance to health care to aviation.

» key insights

- **Long-standing open problems in mathematics can now be solved completely automatically resulting in clever though potentially gigantic proofs.**
- **Our time requires answers to hard questions regarding safety and security. In these cases knowledge is more important than understanding as long as we can trust the answers.**
- **Powerful SAT-solving heuristics facilitate linear speedups even when using thousands of cores. Combined with the ever-increasing capabilities of high-performance computing clusters they enable solving challenging problems.**

ILLUSTRATION BY PETER CROWTHER ASSOCIATES


$(x_1 \vee x_3)$
$(x_1 \vee x_5)$
(x_1)
$d(x_1 \vee x_3)$
$d(x_1 \vee x_5)$
$(\overline{x_3})$
$(\overline{x_5})$

Many problems arising from areas such as Ramsey Theory and formal methods appear to be intrinsically hard and may be only solvable by SAT. Any proof for such problems may be huge, in which case mathematicians will not be able to produce a paper proof. The enormous size of such proofs hardly influences confidence in the correctness, as highly trusted systems can validate them.

We argue that obtaining such results is meaningful regardless of our ability to understand them.

The Rise of Brute Force

We all know that brute force does not work, or at least is brutish, do we not? In our case it is even "brute reasoning."

> ***I can stand brute force, but brute reason is quite unbearable. There is something unfair about its use. It is hitting below the intellect.***
>
> O. Wilde

A mathematician using "brute force" is a kind of barbaric monster, is she not? Case distinctions play an important role for thinking, but if the number of cases gets too big, it seems impossible to obtain an overview, and one has to slavishly follow the details. But perhaps this is what our times demand?

In the beginning of the 20th century there was a very optimistic outlook for mathematics. Gödel's Incompleteness Theorem seemed to destroy the positive spirit of the time, famously expressed by Hilbert's "We must know. We will know." That said, even Gödel anticipated the relevance of SAT solving in his letter to von Neumann[a], shifting the attention to finitizing infinite problems. Today, SAT solving on high-performance computing systems enables us to conquer problems of high complexity, driven by practice. This combination of enormous computational power with "magical brute force" can now solve very hard combinatorial problems, as well as proving safety of systems such as railways.

Our guiding example is the *Pythagorean Triples Problem*,[15,25] a typical problem from Ramsey Theory: we consider all partitions of the set $\{1, 2, \ldots\}$ of natural numbers into finitely many parts, and the question is whether always at least one part contains a Pythagorean triple (a, b, c) with $a^2 + b^2 = c^2$. For example when splitting into odd and even numbers, then the odd part does not contain a Pythagorean triple (due to odd plus odd = even), but the even part contains for example $6^2 + 8^2 = 10^2$. We show that the answer is *yes*,[15] when partitioning into two parts, and we conjecture the answer to be *yes* for any finite size of the partition.

To solve the *Boolean Pythagorean Triples Problem*, it suffices to show the existence of a subset of the natural numbers, such that any partition of that subset into two parts has one part containing a Pythagorean triple. We focus on subsets $\{1, \ldots, n\}$, and determined by SAT solving that the smallest n for which the property holds is 7825. Plain brute force cannot help, since 2^{7825}, the number of possible partitions into two parts, is way too big. So really "clever" algorithms are needed. An interesting aspect here is that there is no known ordinary mathematical existence proof for any form of the Pythagorean Triples Problem, even when generalizing the problem from triples $a^2 + b^2 = c^2$ to tuples $t_1^2 + \cdots + t_{k-1}^2 = t_k^2$. Only computational proofs are known and, so far at least, only SAT solving can deal with the harder problems. We show that $\{1, \ldots, 10^7\}$ can be partitioned into *three* parts, such that *no part* contains a Pythagorean triple. Thus if there is an n such that every 3-partitioning of $\{1, \ldots, n\}$ has a part containing a Pythagorean triple, then $n > 10^7$. Due to this enormous size, it is thus conceivable that the truth of the *three-valued Pythagorean Triples Problem* might never be known.

Before considering the solution process, one may ask, why should we care? Are there problems, for which such reasoning is really useful? Yes, the same techniques are used to prove correctness of hardware and software systems. Finding a bug in a large hardware system is essentially the same as finding a counter-example, and thus is similar to finding a partition avoiding all Pythagorean triples. *Proving* correctness of a system, that is, there is no counter-example, is similar to proving that each partition must contain some Pythagorean triple. SAT solving has revolutionized hardware verification,[5] and now SAT can come to the rescue of mathematics, solving very hard combinatorial problems previously completely out of reach. This collaboration works in both directions, as the applications in mathematics, especially Ramsey Theory, sharpen SAT algorithms: the Cube-and-Conquer method[16] was developed for computing van der Waerden numbers,[1] and recently the Cube-and-Conquer solver TREENGELING[b] won the parallel track of the 2016 SAT competition.[c] Deeper mathematical investigations into the structure of the SAT instances could help with understanding and improving SAT in general.

Well known early mathematical proofs using *Proof by Exhaustion* are the Four-Color Theorem[37] and the proof that no projective plane of order 10 exists.[24] The former is actually a rather small case-distinction by modern standards (only hundreds of cases). The latter invokes a larger, but also man-made case-split (billions of cases), for which it can be determined in advance whether this will succeed. In contrast, we have currently no way of knowing whether the SAT solver's "magic" is sufficient to solve a given problem.

Throughout this article we use the *Boolean Schur Triple Problem* as an example: does there exists a red/blue coloring of the numbers 1 to n, such that there is no monochromatic solution of $a + b = c$ with $a < b < c \leq n$. Compared to the Boolean Pythagorean Triples Problem, all natural numbers are involved, not just square numbers. As a result, there are many more triples, and unsatisfiability is reached much sooner. For $n = 8$ such a coloring exists: color the numbers 1, 2, 4, 8 red and 3, 5, 6, 7 blue. However such a coloring is not possible for $n = 9$. A naive brute-force algorithm would consider all $2^9 = 512$ possible red/blue colorings. We will show that with brute reasoning only six (or even four) red/blue colorings need to be evaluated.

The Art of SAT Solving

A SAT problem uses Boolean variables v (they can be assigned to either `true` or `false`), which are constrained using

a https://rjlipton.wordpress.com/the-gdel-letter/.

b http://fmv.jku.at/lingeling/.

c http://www.satcompetition.org/.

clauses, which are disjunctions of literals x. Literals are either variables $x = v$ or their negations $x = \bar{v}$. A literal x (or $\bar{x}$) is `true` if the corresponding variable v is assigned to `true` (or `false`, respectively). A clause is satisfied if at least one of its literals is assigned to `true`. A SAT formula is a conjunction of clauses. We refer to a solution of a SAT formula as an assignment to its variables that satisfies all its clauses. Formulas with a solution are called *satisfiable*, while formulas without solutions are called *unsatisfiable*. Let $\vee$ and $\wedge$ refer to the logical OR and AND operators, respectively. For example, the formula $(x \vee \bar{y}) \wedge (\bar{x} \vee y)$ with two clauses is satisfiable. The solutions for this formula are the two assignments that assign both x and y to the same value.

SAT solvers, programs that solve SAT formulas, have become extremely powerful over the last two decades. Progress has been by leaps and bounds, starting with the pioneering work by Davis and Putnam until the early 1990s when solvers could handle formulas with thousands of clauses. Today's solvers can handle formulas with millions of clauses. This performance boost resulted in the *SAT revolution*:[3] encode problems arising from many interesting applications as SAT formulas, solve these formulas, and decode the solutions to obtain answers for the original problems. This is in a sense just using the *NP-completeness* of SAT:[6,11,19] every problem with a notion of "solution"—where these solutions are relatively short and where an alleged solution can be verified (or rejected) quickly—can be reduced to SAT efficiently. For many years NP-completeness was used only as a sign of "you can not solve it!", but the SAT revolution has put this back on its feet. For many applications, including hardware and software verification,[7,18] SAT solving has become a disruptive technology that allows problems to be solved faster than by other known means.

The main paradigms of SAT solving are the incomplete *local search*,[20] which can only find satisfying assignments, and the two complete paradigms (which can also determine unsatisfiability), *look-ahead*[17] and *Conflict-Driven Clause Learning*[28] (CDCL). Local search tries to find a solution via local modifications to total assignments (using all variables). Look-ahead recursively splits the problem as cleverly as possible into subproblems, via looking-ahead. CDCL tries to assign variables to find a satisfying assignment in a straight-forward way, and if that fails (the normal case), then the failure is transformed into a clause, which is added to the formula. Here, we first explain CDCL, which is mainly responsible for the SAT revolution. Afterwards we describe how look-ahead can enhance CDCL on hard problems.

CDCL SAT solving algorithms cycle through three phases: *simplify*, *decide*, and *learn*. Solvers maintain an assignment (initially empty) and each phase updates that assignment. During *simplify* the assignment is extended by detecting new inferences. Afterwards, *decide* heuristically picks an unassigned variable and assigns it to `true` or `false`. After iterating these two phases, the current assignment either satisfies the formula, which terminates the search, or falsifies a clause. In the latter case, *learn* this conflict, as a clause, and modify the assignment to resolve the conflict. If the empty clause $\bot$ is learned, the solver detects unsatisfiability, otherwise simplify-decide is performed again, etc. Look-ahead differs from CDCL by using stronger means for *simplify* and *decide*, but weaker means for *learn*.

The most basic inference mechanism in SAT solvers works as follows: a clause is *unit* under an assignment that falsifies all but one of its literals, while leaving the remaining literal unassigned. The only possibility to satisfy a unit clause (under that assignment) is to assign the remaining literal to `true`. A key SAT solving technique is *Unit Clause Propagation* (UCP): Given an assignment and a formula, while the formula has unit clauses, extend the assignment by satisfying the remaining literals in the unit clauses. UCP has two possible terminating states: either all unit clauses have been satisfied, or there is a falsified clause due to two complementary unit clauses (x) and $(\bar{x})$. In the latter case, we say that UCP results in a *conflict*. Conflicts are analyzed to obtain new clauses. These *conflict clauses* are added to the formula to prevent the solver from visiting that assignment in the future. Additionally, conflict analysis updates the heuristics to guide the solver towards a short refutation.

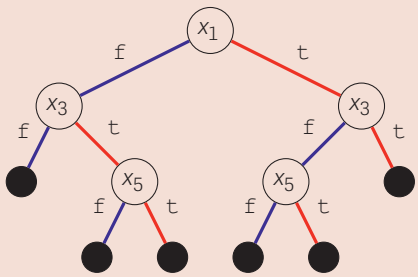
There are two types of decision heuristics for SAT solvers: *focus* and *global* heuristics. Focus heuristics, also known as conflict-driven heuristics (for CDCL solvers), aim at finding short refutations. These heuristics are cheap to compute and have been highly successful in solving large problems arising from industrial applications. In short, focus heuristics work as follows: whenever a solver encounters a conflicting state, the importance of the variables that cause the conflict is increased. Simply making these variables more

Figure 1. Encoding and case split of Boolean Schur Triples Problem.

Encoding

$$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4) \wedge$$
$$(x_1 \vee x_4 \vee x_5) \wedge (\bar{x}_1 \vee \bar{x}_4 \vee \bar{x}_5) \wedge (x_2 \vee x_3 \vee x_5) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_5) \wedge$$
$$(x_1 \vee x_5 \vee x_6) \wedge (\bar{x}_1 \vee \bar{x}_5 \vee \bar{x}_6) \wedge (x_2 \vee x_4 \vee x_6) \wedge (\bar{x}_2 \vee x_4 \vee \bar{x}_6) \wedge$$
$$(x_1 \vee x_6 \vee x_7) \wedge (\bar{x}_1 \vee \bar{x}_6 \vee \bar{x}_7) \wedge (x_2 \vee x_5 \vee x_7) \wedge (\bar{x}_2 \vee \bar{x}_5 \vee \bar{x}_7) \wedge$$
$$(x_3 \vee x_4 \vee x_7) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_7) \wedge (x_1 \vee x_7 \vee x_8) \wedge (\bar{x}_1 \vee \bar{x}_7 \vee \bar{x}_8) \wedge$$
$$(x_2 \vee x_6 \vee x_8) \wedge (\bar{x}_2 \vee \bar{x}_6 \vee \bar{x}_8) \wedge (x_3 \vee x_5 \vee x_8) \wedge (\bar{x}_3 \vee \bar{x}_5 \vee \bar{x}_8) \wedge$$
$$(x_1 \vee x_8 \vee x_9) \wedge (\bar{x}_1 \vee \bar{x}_8 \vee \bar{x}_9) \wedge (x_2 \vee x_7 \vee x_9) \wedge (\bar{x}_2 \vee \bar{x}_7 \vee \bar{x}_9) \wedge$$
$$(x_3 \vee x_6 \vee x_9) \wedge (\bar{x}_3 \vee \bar{x}_6 \vee \bar{x}_9) \wedge (x_4 \vee x_5 \vee x_9) \wedge (\bar{x}_4 \vee \bar{x}_5 \vee \bar{x}_9)$$

Case split as binary tree



important than all the other variables results in state-of-the-art performance on most industrial problems.[2]

If no short refutation exists (or is too hard to find), it is best to use global heuristics (for look-ahead solvers) to split the search space into two parts that are both easier to solve. Global heuristics are based on *look-aheads*:[23] for a given formula F, a look-ahead on literal x assigns x to `true`, applies UCP, and computes the set S of clauses in F that are shortened, but not satisfied. The heuristic value of a look-ahead on x is based on a weighted sum of the clauses in S, where clause weights depend on the length of clauses.

Both focus and global heuristics can reduce the search space exponentially. For really hard problems, such as the Pythagorean Triples Problem, it is best to combine both types of heuristics. Focus heuristics are effective when there exists a short refutation of the formula. For hard problems, initially there are no short refutations. One therefore needs to partition such a problem using global heuristics until the short refutations manifest themselves. This is the main idea behind the Cube-and-Conquer SAT solving paradigm,[16] which was crucial to solve the Pythagorean Triples Problem.

Consider again the Boolean Schur Triples Problem on the existence of a red/blue coloring of 1, …, 9 without a monochromatic solution of $a + b = c$. Figure 1 shows the SAT encoding, consisting of 32 clauses using the Boolean variables $x_1, \ldots, x_9$. If variable x_i is assigned to `true` (`false`), then number i is colored red (blue). For each of the 16 solutions of $a + b = c$, there are two clauses: one stating that at least one of a, b, or c must be colored red, one stating that at least one of them must be colored blue. A binary tree is shown right beside the clauses. Each internal node contains a splitting variable x_i. The left branches assign decision variables to `false` (blue edge), while the right branches assign decision variables to `true` (red edge). Each leaf node represents an assignment that would result in a conflict during UCP. For example, for the left-most leaf node, x_1 and x_3 are assigned to `false` (blue): thus x_2, x_4 have to be set to `true` (due to $1 + 2 = 3$ and $1 + 3 = 4$), forcing x_6 to `false` ($2 + 4 = 6$), which forces x_7 and x_9 to `true` ($1 + 6 = 7$ and $3 + 6 = 9$), which yields the conflict $2 + 7 = 9$ with all three set to `true` (red). This node matches the first clause in the proof of Figure 2. The binary tree (a simple form of look-ahead solving) illustrates that heuristics can reduce the number of assignments to be evaluated from 512 to 6.

Due to the limited size of the example formula, relatively simple heuristics are sufficient to reduce the number of cases from 512 to 6. One such simple heuristic is Maximum Occurrences in clauses of Minimal Size (MOMS). Initially, all clauses are ternary and variable x_1 occurs most frequently. Therefore x_1 is used as the first decision variable. After simplification, several variables occur most frequently in binary clauses (twice), but variable x_3 has the best tie break (occurrences in remaining ternary clauses). Therefore variable x_3 is the best decision on the second level of the tree. Finally, variable x_5 is the most occurring variable in binary clauses on the third level.

A crucial aspect of solving the Boolean Pythagorean Triples Problem was the use of a dedicated look-ahead heuristic based on the recursive weight heuristic for random 3-SAT formulas. The three magic constants in this heuristic have been manually tweaked to achieve strong performance on the Boolean Pythagorean Triples Problem.[15] We estimate that the use of this optimized look-ahead heuristic reduced the number of cases by at least two orders of magnitude compared to alternative heuristics, such as focus heuristics or MOMS. Look-ahead heuristics were popular in the 1990s, but they have been mostly ignored after CDCL emerged. The usefulness of look-ahead heuristics to boost the performance on hard problems may revive the interest.

Proofs of Unsatisfiability

The unpredictable effectiveness of SAT solvers, together with their non-trivial implementations (needed for real-world efficiency), raise the question of whether their results can be trusted. If a problem has a solution, it is easy to verify that the given solution is correct: simply check whether the solution satisfies at least one literal in every clause. However, a claim that no solution exists is much harder to validate. Since SAT solvers use many complicated techniques that could result in implementation as well as conceptual errors, a method is required to verify unsatisfiability claims.

There are two approaches to deal with the trust issue of complicated software: prove its correctness or produce a certificate which can be validated with a simple program. Work in the first direction resulted in verified SAT solving.[31] However, this approach has two disadvantages: only some state-of-the-art techniques are verified, and verification is performed only on "higher levels," and thus excludes the low-level implementation tricks that

Figure 2. Proof and unit clause justification of the Boolean Schur Triples Problem.

Proof	Unit clause justification
$(x_1 \vee x_3)$	$(\cancel{x_1} \vee x_2 \vee \cancel{x_3})$, $(\cancel{x_1} \vee \cancel{x_3} \vee x_4)$, $(\cancel{\bar{x}_2} \vee \cancel{\bar{x}_4} \vee \bar{x}_6)$, $(\cancel{x_1} \vee \cancel{x_6} \vee x_7)$, $(\cancel{x_3} \vee \cancel{x_6} \vee x_9)$, $(\cancel{\bar{x}_2} \vee \cancel{\bar{x}_7} \vee \cancel{\bar{x}_9})$
$(x_1 \vee x_5)$	$(\cancel{x_1} \vee x_3)$, $(\cancel{x_1} \vee x_4 \vee \cancel{x_5})$, $(\cancel{x_1} \vee \cancel{x_5} \vee x_6)$, $(\bar{x}_2 \vee \cancel{\bar{x}_4} \vee \cancel{\bar{x}_6})$, $(\cancel{x_2} \vee \cancel{x_5} \vee x_7)$, $(\cancel{\bar{x}_3} \vee \cancel{\bar{x}_4} \vee \cancel{\bar{x}_7})$
(x_1)	$(\cancel{x_1} \vee x_3)$, $(\cancel{x_1} \vee x_5)$, $(\bar{x}_2 \vee \cancel{\bar{x}_3} \vee \cancel{\bar{x}_5})$, $(\cancel{\bar{x}_3} \vee \cancel{\bar{x}_5} \vee \bar{x}_8)$, $(\cancel{x_2} \vee x_6 \vee \cancel{x_8})$, $(\cancel{x_1} \vee \cancel{x_8} \vee x_9)$, $(\cancel{\bar{x}_3} \vee \cancel{\bar{x}_6} \vee \cancel{\bar{x}_9})$
$d(x_1 \vee x_3)$	
$d(x_1 \vee x_5)$	
$(\bar{x}_3)$	(x_1), $(\cancel{\bar{x}_1} \vee \bar{x}_2 \vee \cancel{\bar{x}_3})$, $(\cancel{\bar{x}_1} \vee \cancel{\bar{x}_3} \vee \bar{x}_4)$, $(\cancel{x_2} \vee \cancel{x_4} \vee x_6)$, $(\cancel{\bar{x}_1} \vee \cancel{\bar{x}_6} \vee \bar{x}_7)$, $(\cancel{\bar{x}_3} \vee \cancel{\bar{x}_6} \vee \bar{x}_9)$, $(\cancel{x_2} \vee \cancel{x_7} \vee \cancel{x_9})$
$(\bar{x}_5)$	(x_1), $(\bar{x}_3)$, $(\cancel{\bar{x}_1} \vee \bar{x}_4 \vee \cancel{\bar{x}_5})$, $(\cancel{\bar{x}_1} \vee \cancel{\bar{x}_5} \vee \bar{x}_6)$, $(x_2 \vee \cancel{x_4} \vee \cancel{x_6})$, $(\cancel{\bar{x}_2} \vee \cancel{\bar{x}_5} \vee \bar{x}_7)$, $(\cancel{x_3} \vee \cancel{x_4} \vee \cancel{x_7})$
$\bot$	(x_1), $(\bar{x}_3)$, $(\bar{x}_5)$, $(x_2 \vee \cancel{x_3} \vee \cancel{x_5})$, $(\cancel{x_3} \vee \cancel{x_5} \vee x_8)$, $(\cancel{\bar{x}_2} \vee \bar{x}_6 \vee \cancel{\bar{x}_8})$, $(\cancel{\bar{x}_1} \vee \cancel{\bar{x}_8} \vee \bar{x}_9)$, $(\cancel{x_3} \vee \cancel{x_6} \vee \cancel{x_9})$

are crucial for fast performance. Both disadvantages slow down the verified solver substantially, making it useless in most practical settings.

The second approach has been more successful in the context of SAT solving. We refer to a certificate of an unsatisfiability claim as a *proof of unsatisfiability*. What kind of format would be useful for such proofs? The ideal proof format facilitates five properties: (1) proof production should be *easy* to ensure that it will be supported by many solvers; (2) proofs should be *compact* in order to have small overhead; (3) proof validation should be *simple*, otherwise the trust issue persists; (4) proof validation should be *efficient* to make verification useful in practice; and (5) all techniques should be *expressible*, otherwise solvers will be handicapped. There is a trade-off between these properties. For example, more details in a proof should allow a more efficient validation procedure. However, adding details makes proofs less compact and harder to produce.

Initially, proofs of unsatisfiability were based on resolution. Although useful in some settings, it is hard or even impossible to achieve the properties of easy production (1), compactness (2), and expressibility (5) for such proofs. The alternative is *clausal proofs*[12] for which it is now possible to achieve all five properties.

What is a clausal proof of unsatisfiability for a SAT problem? Basically, we start with the given list of clauses, and add or delete clauses, until finally we add the empty clause $\bot$, which marks unsatisfiability, since there is no literal in it to satisfy. The most basic restriction on adding clauses is, that the addition is *solutions-preserving*, that is, all solutions (at that point, taking all previous additions and deletions into account) also satisfy the added clause. This guarantees correctness: if all additions are solutions-preserving, and we are able to add $\bot$ (which has no solution), then the original SAT problem must be unsatisfiable. For example, consider the formula $F = (x \vee y) \wedge (x \vee \bar{y})$. Adding the clause (x) to F is solutions-preserving: F has two solutions and in both solutions x is assigned to `true`.

It is important to validate that clause addition steps are solutions-preserving, otherwise we do not have a *proof*, just some sort of claim. This verification should be cheap to perform, and the basic criterion is as follows. Suppose a formula F is given, and the clause C is claimed to be solutions-preserving for F. Take the assignment that sets all literals in C to `false`. If UCP on F results in a conflict, then the clause is indeed solutions-preserving, since we checked that it is not possible to falsify C while satisfying F. This realizes the first three ideal proof format properties: easy, compact, and simple. The solver can just output the learned clauses, without a justification, and validation happens by UCP.

SAT solvers do not only learn lots of clauses, but also aggressively delete them to achieve fast UCP. Proofs should include this deletion information in order to realize efficient validation. Furthermore, proof checkers require dedicated UCP algorithms to make proof validation as fast as proof production.[14] Combining these techniques realizes the fourth ideal proof property (efficient validation).

A proof of our running example is shown in Figure 2. The proof consists of six clause addition steps and two clause deletion steps. The latter have a "d" prefix and do not require checking. The correctness of each clause addition step is checked using UCP, and shown using a unit clause justification: a sequence of clauses that become unit, ending with a falsified clause that marks the conflict. The unit clause justification is omitted from clausal proofs to ensure compactness, but the checker constructs a justification during validation.

Some SAT solving techniques may change (add or remove) solutions which can significantly reduce solving time. In order to express such techniques—to have also the final ideal proof property (expressible)—support is required for proof steps that go beyond the above solutions-preservation. This is realized by the concept of *solutions-preserving modulo x* for some literal x. Let φ be an assignment. We denote by $\varphi \oplus x$ the assignment obtained by flipping the truth value for literal x in φ. In case x is unassigned in φ, then x is assigned to `true` in $\varphi \oplus x$. For a given formula F, addition of clause C is solutions-preserving modulo x if for all solutions φ of F at least one of φ or $\varphi \oplus x$ satisfies F *and* C.

For example, consider the formula $F = (x \vee y) \wedge (x \vee \bar{y})$ again. The addition of clause $(\bar{x} \vee y)$ to F is solutions-preserving modulo y. Recall that F has two solutions. The first solution φ_1, where x is `true` and y is `true`, also satisfies $(\bar{x} \vee y)$. The second solution φ_2, where x is `true` and y is `false`, falsifies $(\bar{x} \vee y)$, but $\varphi_2 \oplus y$ satisfies F and $(\bar{x} \vee y)$.

How to check that adding clause C is solutions-preserving modulo x? We use the following efficient criterion: $x \in C$, and for all $D \in F$ with $\bar{x} \in D$ we have that setting all literals in C *as well as* all literals in $D \backslash \{\bar{x}\}$ to `false` yields a conflict via UCP. The proof format that encapsulates this inference in a single step is called the "DRAT" format,[4] and is supported by state-of-the-art solvers.

It is instructive to show that this criterion guarantees adding C to F is solutions-preserving modulo x. The critical clauses are the $D \in F$ with $\bar{x} \in D$, since here flipping of x might change a satisfied clause to a falsified clause. First observe that from the criterion follows that all $C \cup (D \backslash \{\bar{x}\})$ are solutions-preserving w.r.t. F. Now assume that φ is a total satisfying assignment for F which falsifies C (otherwise φ satisfies F and C and we are done). Thus φ falsifies x, and $\varphi \oplus x$ satisfies C. Since all $C \cup D \backslash \{\bar{x}\}$ are solutions-preserving w.r.t. F, φ satisfies all $C \cup D \backslash \{\bar{x}\}$. Hence φ satisfies all $D \backslash \{\bar{x}\}$ (because φ falsifies C), and so does $\varphi \oplus x$ as well, and thus indeed $\varphi \oplus x$ satisfies all D. QED

The DRAT format seems to be a good proof format for existing and future SAT solvers, as it has all the five properties of an ideal proof format. Moreover, DRAT proofs can be efficiently checked even in parallel, and they have been used to validate the results of the annual international SAT competitions since 2013. For the Boolean Schur Triples Problem with $n = 9$, there exists a DRAT proof consisting of only four clause additions: $(x_1 \vee x_4), (x_1), (x_4), \bot$. Validating this proof involves more details, which can be obtained by using the DRAT proof checker DRAT-TRIM.[d]

Indeed, DRAT in a theoretical sense is equivalent to one of the most powerful systems studied in proof complexity, Extended Frege with Substitution, and thus it should offer "proofs as short as possible."[4] The Extension

d The tool is available at https://github.com/marijnheule/drat-trim.

Rule basically states that the clauses $(x \vee \bar{a} \vee \bar{b}) \wedge (\bar{x} \vee a) \wedge (\bar{x} \vee b)$ can be added if no literals x and $\bar{x}$ occur in the formula. In fact, each of the clauses are solutions-preserving modulo x or $\bar{x}$ according to the above criterion.

Proof size nevertheless becomes an issue. Although DRAT proofs are "compact," the size of the DRAT proof of the Boolean Pythagorean Triples Problem is 200 TB. An obvious challenge of such a huge file is its storage. Also, dealing with such files increases the complexity of proof validation algorithms, which will need to support parallel checking. On the other hand, it is possible to trade complexity for space by adding details to the proof that facilitate fast checking. In order to make this feasible, the proof can be optimized using a non-verified trimmer which also adds the checking details. This approach has been successfully applied to validate the 200 TB proof using a checker which was *formally verified* in Coq.[8]

Ramsey Theory and Complexity

A popularized summary of Ramsey Theory is that "complete chaos is impossible."[26] More concretely, Ramsey Theory deals with patterns that occur in well-known sets such as the set of natural numbers or the set of graphs. For example, coloring the natural numbers with finitely many colors will result in a monochromatic Schur triple $a + b = c$.

Hundreds of papers have been published on determining the smallest size of sets such that a given pattern must start to occur.[32] The most famous pattern is related to Ramsey numbers $R(k)$: the smallest n such that all red/blue edge colorings of the complete graph with n vertices have a red or a blue clique of size k. Only the first four Ramsey numbers are known. Paul Erdős famously told a story about aliens who threatened to obliterate earth unless humans provided them with the value of $R(5)$—with a proof, we may add here. Putting all mankind behind this project would do the job in a year. Yet if aliens asked for $R(6)$, we should opt for the Hollywood resolution and obliterate them instead.[13]

Many problems in Ramsey Theory appear to be solved only using large case splits (especially for the determination of Ramsey-type numbers), and thus using SAT is a natural option. Also SAT formulations of these problems are easy and natural. In order to determine the smallest subset in which a pattern starts to occur using SAT, two formulas need to be solved. First, it has to be shown that for any smaller subset there exists a counter-example. This is typically easy, because the formula is satisfiable. The second formula, encoding the existence of the pattern, is much harder to solve as now unsatisfiability must be shown.

The first major success of SAT solving in Ramsey Theory was determining the sixth Boolean van der Waerden number:[22] vdW(6) = 1132. The number vdW(k) expresses the smallest n such that any red/blue coloring of the numbers 1 to n results in a monochromatic arithmetic progression of length k. The computation used multiple clusters as well as dedicated SAT-solving hardware (FPGA solvers) for several months. Unfortunately, no proof was produced during the computation, making it impossible to verify the result. This raises several trust issues, because errors could have been made on several levels. For example, was the splitting correct and thus has the whole search space been explored? Also, FPGA solvers have been tested much less thoroughly compared to state-of-the-art solvers.

The first important problem with a verified clausal proof is the Erdős Discrepancy Problem (EDP), which is about "complete uniformity is impossible." The problem conjectures that any infinite sequence $s_1, s_2, \ldots$ with $s_i = \pm 1$ contains for any positive integer C a subsequence $s_d, s_{2d}, s_{3d}, \ldots, s_{kd}$, for some positive integers k and d, such that $\left|\sum_{i=1}^{k} s_{id}\right| \geq C$. Using colors, the conjecture says that for every $C \geq 1$ and every red/blue coloring of $1, 2, \ldots$ there is a finite initial segment of some progression $d, 2d, 3d, \ldots$ for some $d \geq 1$, such that the discrepancy between the number of color-occurrences is at least C (one color occurs at least C-times more than the other). The conjecture has been a long-standing open problem even for $C = 2$. The case $C = 2$ was eventually solved using SAT by providing the exact bound,[21] also applying Cube-and-Conquer. The encoding of this problem is more involved than the simple encoding of Ramsey problems (which are just hypergraph coloring problems), and thus, though a clausal proof has been provided, correctness is more of an issue than in cases of Ramsey Theory. Computationally, EDP is much easier,[21] and a much smaller proof exists (about a gigabyte) than in our case. Finally a general mathematical existence proof has been provided.[35] This mathematical proof was called "much more satisfying" than the computational approach.[25] However, there is for example the possibility that the Pythagorean Tuples Conjecture (see below) is not provable with current methods. Furthermore, the SAT approach is actually a rather "satisfying approach" when taking into account its deep connections to formal methods.

The *Pythagorean Tuples Conjecture* states that Ptn(k; m)—with k the length of the tuple and m the number of colors—exists for all $k \geq 3$ and $m \geq 2$. That is, for every partitioning of $\{1, \ldots, \text{Ptn}(k; m)\}$ into m parts, some part contains a Pythagorean tuple of size k. We have shown that Ptn(3; 2) = 7825. The value of Ptn(3; 2) was conjectured[30] not to exist after determining the numbers Ptn(k; 2) for $4 \leq k \leq 31$. We have meanwhile computed the only known Pythagorean tuples numbers for three colors: Ptn(5; 3) = 191, Ptn(6; 3) = 121, and Ptn(7; 3) = 102. We also established Ptn(3; 3) > 10^7, and this lower bound (via local-search algorithms) seems still far away from the exact bound. So it is imaginable that a mathematical existence-proof can not be found, and finiteness of Ptn(3; 3) might never be established. It is furthermore conceivable that the Pythagorean Tuples Conjecture is true but the best proofs are SAT-like. Thus formal proofs in systems like Zermelo-Fraenkel set theory would only *exist* for concrete k and m, while there would not exist a single proof for all k and m. No mathematical existence proofs have yet been established for any Ptn(k; m) (see "alien truth statements" for further discussions).

Before coming to the industrial applications of SAT, we remark that the Ramsey numbers[33] $R(k)$ are very different from the Boolean Pythagorean Triples Problem: namely the latter is "random-like" and thus has no symmetries (besides the trivial color symmetries). Currently SAT solving is more successful in the absence of strong symmetries, while Ramsey numbers currently have too much structure for an automated attack. More sophisticated

symmetry-breaking techniques are required to improve the performance.

Brute Force Formal Methods

SAT solvers are a key technology in formal methods for applications, such as bounded model checking[5] and equivalence checking. In bounded model checking, given a transition system and an invariant such as a safety property, the SAT solver determines for some appropriate finitization, whether there exists a sequence of transitions that violates the safety property. Equivalence checking is used to determine the equivalence of a specification and an implementation or two different implementations. The SAT solver is asked to find an input such that some output differs. Notice that the existence of a solution means that the safety property is violated or that there exists a counter-example for equivalence.

All problems discussed so far could be expressed as a propositional formula. For many interesting problems, however, this is not the case and they require a richer logic for its representation. That does not mean that SAT technology cannot be used to solve these problems. On the contrary: more and more problems that require a richer logic are being solved efficiently using SAT.

The key idea is to abstract away those parts of a given problem that cannot be expressed as propositional logic. A solution of the abstracted problem may not be a solution of the given problem, while a refutation of the abstracted problem is also a refutation of the given problem. In case a solution of the abstracted problem is obtained, which is not a solution for the given problem, then the abstraction is refined by adding a clause that prevents the SAT solver from finding that solution (and potentially similar solutions) again. This is repeated until either a refutation or a solution for the given problem is found. Incremental SAT solving[10] facilitates an efficient implementation of this approach.

This approach has been very successful in Automated Theorem Proving (ATP). The long-time champion in the annual ATP competitions is VAMPIRE,[36] which has been tightly integrated with a SAT solver. Other strong ATP solvers, including iPROVER and LEO, incorporate SAT solvers as well. The major interactive theorem provers, such as ACL2, COQ, and ISABELLE, support the usage of SAT solvers to deal with subproblems that can be expressed in propositional logic. In this setting, SAT solvers are treated as a black-box and the emitted proofs are validated in the theorem provers. Another successful extension of SAT in this direction is *Satisfiability Modulo Theories* (SMT).[9] It uses multiple theories, such as linear arithmetic, uninterpreted functions, and bit-vectors, and replaces constraints in a theory by propositional variables. SMT solvers, such as Z3, BOOLECTOR, CVC4, and YICES have been highly successful.

More and more problems that require a richer logic are being solved efficiently using SAT.

Alien Truths

The core argument against solving a problem by brute force is it does not contribute to understanding the problem. In that view, the proof is meaningless and hard to generalize, and a human mathematical proof is preferred. Furthermore, without understanding errors seem more likely, although validation can be done by highly trusted systems.

The proponents of "elegant" proofs appear to consider problems with only very long proofs as not interesting or not relevant. But even unprovable statements, like the famous Continuum Hypothesis, have an important place in mathematics. If we do not study the limits of our current knowledge, we will stay ignorant forever, always restricted to a "safe space," neglecting problems we *assume* to be too hard. Furthermore, what is a limit of one discipline is a core subject of another discipline. Computational complexity and Ramsey Theory have close relations. *Understanding* the hardness of problems from Ramsey instances could lead to major breakthroughs.[27] For example, *why* is proving the Ramsey property for $a + b = c$ rather easy, while $a^2 + b^2 = c^2$ appears to be a very hard problem? In general, even small propositional problems might have only very large proofs. If we would ignore this area, then we would allow random holes in our knowledge. The question "why there are *no* short proofs," and "what makes a problem hard," are deep and fascinating questions, and we consider them some of the most important problems of our times.

To better discuss the untold stories of computer science, complexity

theory, and SAT, let's call *alien* a provable and rather short mathematical statement with only a very long proof. Artificial alien statements can be constructed using Gödel's methods. Whether a natural truth statement can be shown to be alien, such as the Pythagorean Triples Problem, is of highest relevance. Even if a short proof for the Pythagorean Triples Problem may be constructed, that is unlikely to be the case for the exact bound result. Now there is actually a whole spectrum of possibilities between human truths and alien truths. Classical mathematical statements for which a paper proof exists, such as Schur's Theorem,[34] we consider as *human* truth statements. Hence the vast body of mathematical works belongs to this category. Furthermore, we consider mathematical statements that have been proven mostly manually, but with some computer help, *weakly human*. More specifically, such statements have a large case-split, which could potentially be understood by humans, but which have only been checked mechanically. An example of such a statement is the Four-Color Theorem.[37] The proof by Appel and Haken considers 663 cases in its improved version. The case-split is fully understood and humanly constructed. A theorem prover only checks the cases. Coming to larger cases, we refer to a *weakly alien* truth statement as a giant humanly generated case-split which can be validated using plain brute-force methods. For example, it has been shown that the minimum number of givens is 17 in Sudoku by enumerating all possible cases with 16 givens and refuting them all[29] (5 472 730 538 cases after symmetry breaking). Although impossible to evaluate by humans, it could be directly done mechanically. This result is expected to be weakly alien, as it is unlikely that there exists a small enough case-split that is checkable by humans.

We arrive at a better understanding of "alien," namely a truth statement is *alien* if humanly understandable case-splits are way too big for any plain brute-force method, but there exists a giant case-split that mysteriously avoids an enormous exponential effort. Examples of truth statements that are expected to be alien are that vdW(6) = 1132 (see Kouril[22]) and that the exact bound of EDP with $C = 2$ is 1161 (see Konev[21]). A plain brute-force approach to those problems would require the evaluation of 2^{1132} and 2^{1161} cases, respectively. Brute reasoning using SAT solvers significantly reduced the size of the case-splits and allowed determining their truth. We think it is relevant to make a further distinction: the above two alien truth statements express the exact bound, but for both cases there is a mathematical existence proof that the pattern cannot be avoided indefinitely. Now also high-level statements, such as any red/blue coloring of the natural numbers yields a monochromatic Pythagorean triple, could be alien, when the bound result, Ptn(3; 2) = 7825, is the only proof. We call such statements indeed *strongly alien*. If a mathematical existence proof would be found for the statement here, then only the bound statement remains, which is simply *alien*. This happened for the Erdős Discrepancy Problem: the bound was computed using SAT, and later a mathematical existence proof was given.

For some truth statements, we may never be able to produce a proof.

Finally, for some truth statements, we may never be able to produce a proof. A possible example problem of this type is the statement that every 3-coloring of the natural numbers yields a monochromatic Pythagorean triple. As already discussed, experiments show that $\text{Ptn}(3; 3) > 10^7$, where lower bounds are relatively easy to compute. Proofs of upper bound results are much harder to obtain: for example, $\text{Ptn}(3; 2) > 7824$ can be computed in one CPU-minute with local search, while computing $\text{Ptn}(3; 2) \leq 7825$ required more than 40 000 CPU-hours. We call decidable truth statements *extra alien* if a proof can never be computed.

The concept of alien truth statements deals with the *size* of proofs, but it touches naturally on *unprovability* (in current systems like Zermelo-Fraenkel set theory). It is conceivable that Ptn(3; 3) does not exist, that is, the natural numbers are 3-colorable without a monochromatic Pythagorean triple. However, this may not be provable, since the coloring is too complex. On the other hand, it is conceivable that all Ptn(3; m) with $m \geq 3$ exist (note that a SAT solver can prove them in

principle), but these statements are all alien or extra-alien. Since these proofs grow with m, the general statement that all Ptn(3; m) with $m \geq 3$ exist, is then unprovable *in principle*.

Conclusion

Recent successes in brute reasoning, such as solving the Erdős Discrepency Problem and the Pythagorean Triples Problem, show the potential of this approach to deal with long-standing open mathematical problems. Moreover, proofs for these problems can be produced and verified completely automatically. These proofs may be big, but we argued that compact elegant proofs may not exist for some of these problems, in particular (but not only) for the exact bound results. The size of these proofs does not influence the level of correctness, and these proofs may reveal interesting information about the problem.

In contrast to popular belief, mechanically produced huge proofs can actually help in understanding the given problem. We can try to understand their structure, and making them thus smaller. Hardly any research has been done yet in this direction apart from removing redundancy in a given proof. Possibilities are changing the heuristics of a solver or introducing new definitions of frequently occurring patterns in the proof. Indeed, simply validating a clausal proof does not only produce a yes/no answer as to whether the proof is correct, but also provides an *unsatisfiable core* consisting of all original clauses that were used to validate the proof—revealing important parts of the problem. The size of the core depends on the type of problem. Problems in Ramsey Theory typically have quite a large core and therefore provide limited insight. Many bounded model checking problems, however, have small unsatisfiable cores, thereby showing that large parts of the hardware design were not required to determine the safety property.

To conclude, it is definitely possible to gain insights by using SAT. However that "insight" might need to be reinterpreted here, and might work on a higher level of abstraction. Every paradigm change means asking different questions. Gödel's Incompleteness Theorem *solved* partially the question of the consistency of mathematics by showing that the answer provably cannot be delivered in the näive way. Now the task is to live up to big complexities, and to embrace the new possibilities. Proofs must become objects for investigations, and understanding will be raised to the next level, how to find and handle them.

So, when the day finally comes and the aliens arrive and ask us about Ptn(3; 3), we will tell them: "You know what? Finding the answer yourself gives you a much deeper understanding than just telling you the answer—here you have the SAT solving methodology, that's the real stuff!" And then humans and aliens will live happily ever after.

> Wir müssen wissen. Wir werden wissen.
> (We must know. We will know.)
> David Hilbert, 1930 C

References

1. Ahmed, T., Kullmann, O., Snevily, H. On the van der Waerden numbers w(2; 3, *t*). *Disc. Appl. Math. 174* (2014), 27–51.
2. Biere, A., Fröhlich, A. Evaluating CDCL variable scoring schemes. In *SAT* (Springer, 2015), 405–422.
3. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. eds. *Handbook of Satisfiability*, volume 185 of *FAIA*. IOS Press, Amsterdam, The Netherlands, Feb. 2009.
4. Buss, S. Propositional proofs in Frege and Extended Frege systems (abstract). In *Computer Science—Theory and Applications* (Springer, 2015), 1–6.
5. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y. Bounded model checking using satisfiability solving. *Formal Methods in System Design 19*, 1 (2001), 7–34.
6. Cook, S.A. The complexity of theorem-proving procedures. In *STOC* (1971), 151–158.
7. Copty, F., Fix, L., Fraer, R., Giunchiglia, E., Kamhi, G., Tacchella, A., Vardi, M.Y. Benefits of bounded model checking at an industrial setting. In *CAV* (Springer, 2001), 436–453.
8. Cruz-Filipe, L., Marques-Silva, J.P., Schneider-Kamp, P. Efficient certified resolution proof checking, 2016. https://arxiv.org/abs/1610.06984.
9. de Moura, L., Bjørner, N. Satisfiability modulo theories: Introduction and applications. *Communications of the ACM 54*, 9 (2011), 69–77.
10. Eén, N., Sörensson, N. Temporal induction by incremental SAT solving. *Electr. Notes Theor. Comput. Sci. 89,* 4 (2003), 543–560.
11. Garey, M.R., Johnson, D.S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
12. Goldberg, E.I., Novikov, Y. Verification of proofs of unsatisfiability for CNF formulas. In *DATE* (IEEE, 2003), 10886–10891.
13. Graham, R.L., Spencer, J.H. Ramsey theory. *Scientific American 263*, 1 (July 1990), 112–117.
14. Heule, M.J.H., Hunt, W.A. Jr., Wetzler, N. Trimming while checking clausal proofs. In *FMCAD* (IEEE, 2013), 181–188.
15. Heule, M.J.H., Kullmann, O., Marek, V.W. Solving and verifying the Boolean Pythagorean Triples problem via Cube-and-Conquer. In *SAT* (Springer, 2016) 228–245.
16. Heule, M.J.H., Kullmann, O., Wieringa, S., Biere, A. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In *HVC* (Springer, 2011), 50–65.
17. Heule, M.J.H., van Maaren, H. Look-ahead based SAT solvers. In Biere et al. [3], Chapter 5, (2009), 155–184.
18. Ivančić, F., Yang, Z., Ganai, M.K., Gupta, A., Ashar, P. Efficient SAT-based bounded model checking for software verification. *Theoretical Computer Science 404*, 3 (2008), 256–274.
19. Karp, R.M. Reducibility among combinatorial problems. In *Complexity of Computer Computations* (Plenum Press, 1972), 85–103.
20. Kautz, H.A., Sabharwal, A., Selman, B. Incomplete algorithms. In Biere et al. [3], Chapter 6, (2009), 185–203.
21. Konev, B., Lisitsa, A. Computer-aided proof of Erdös discrepancy properties. *Artificial Intelligence 224*, C (July 2015), 103–118.
22. Kouril, M., Paul, J.L. The van der Waerden number *W*(2, 6) is 1132. *Experimental Mathematics 17*, 1 (2008), 53–61.
23. Kullmann, O. Fundaments of branching heuristics. In Biere et al. [3], Chapter 7, (2009), 205–244.
24. Lam, C.W.H. The search for a finite projective plane of order 10. *The American Mathematical Monthly 98*, 4 (April 1991), 305–318.
25. Lamb, E. Maths proof smashes size record: Supercomputer produces a 200-terabyte proof—but is it really mathematics? *Nature 534* (June 2016), 17–18.
26. Landman, B.M., Robertson, A. *Ramsey Theory on the Integers*, volume 24 of *Student mathematical library*. American Mathematical Society, Providence, RI, 2003.
27. Lauria, M., Pudlák, P., Rödl, V., Thapen, N. The complexity of proving that a graph is Ramsey. In *ICALP* (Springer, 2013), 684–695.
28. Marques-Silva, J.P., Lynce, I., Malik, S. Conflict-driven clause learning SAT solvers. In Biere et al. [3], Chapter 4, (2009), 131–153.
29. McGuire, G., Tugemann, B., Civario, G. There is no 16-clue Sudoku: Solving the Sudoku minimum number of clues problem via hitting set enumeration. *Experimental Mathematics 23*, 2 (2014), 190–217.
30. Myers, K.J. *Computational advances in Rado numbers*. PhD thesis, Rutgers University, New Brunswick, NJ, 2015.
31. Oe, D., Stump, A., Oliver, C., Clancy, K. versat: A verified modern SAT solver. In *VMCAI* (Springer, 2012) 363–378.
32. Radziszowski, S.P. Small Ramsey numbers. *The Electronic Journal of Combinatorics* (January 2014), Dynamic Surveys DS1, Revision 14.
33. Ramsey, F.P. On a problem of formal logic. *Proceedings of the London Mathematical Society 30* (1930), 264–286.
34. Schur, I. Über die Kongruenz $x^m + y^m = z^m \pmod p$. *Jahresbericht der Deutschen Mathematiker-Vereinigung 25* (1917), 114–116.
35. Tao, T. The Erdös discrepancy problem. *Discrete Analysis 1* (February 2016), 29.
36. Voronkov, A. AVATAR: The architecture for first-order theorem provers. In *CAV* (Springer, 2014) 696–710.
37. Wilson, R. *Four Colors Suffice: How the Map Problem Was Solved*. Princeton University Press, Princeton, NJ, revised edition, 2013.

Marijn J.H. Heule (marijn@cs.utexas.edu) is a research scientist at The University of Texas, Austin.

Oliver Kullmann (o.kullmann@swansea.ac.uk) is an associate professor in computer science at Swansea University, U.K.

Watch the authors discuss their work in this exclusive *Communications* video. https://cacm.acm.org/videos/the-science-of-brute-force